UCRL- 84015
PREPRINT

Raster Generation in the Array Processor: Trials and Traumas

Neil Maron
Tom A. Brengle

This paper was prepared for submittal to the 1980 Fourth Annual FPS Users' Conference, San Francisco, California April 28-30, 1980

February 25, 1980

Lawrence Livermore Laboratory

Raster Generation in the Array Processor: Trials and Traumas

Neil Maron and Tom A. Brengle[*]
Lawrence Livermore Laboratory
Livermore, California 94550

ABSTRACT

The purpose of this paper is to discuss the methods we developed to handle bit manipulation on the FPS Array Processor 38 bit main data memory and to discuss the problems and successes in converting vector graphics data to raster data.

INTRODUCTION

Previous measurements demonstrate the FPS Array Processor (AP) to be six times faster than a PDPKI-10. This indicates that the AP may be a powerful adjunct to the PDP-10 in processes which are not strictly floating point number oriented. One such application given consideration is that of converting vector data graphics files to compressed raster data files for subsequent plotting on a printer/plotter which is connected to the PDP-10. The conversion to a raster requires a 1024 by 1024 bit buffer if one is using the brute force method. Other methods involve sorting the vectors but require more CPU cycles to perform the sort. If one has sufficient memory then the brute force technique is asserted to be faster. In our configuration the AP has 64K of slow main data memory (333ns MD).

We already have an existing PDP-10 program to do this conversion to raster and then compression for subsequent plotting. This program is slow for several reasons. One of which is the PDP-10 has a virtual

[*]Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore Laboratory under contract number W-7405-ENG-48.

memory operating system on it. In a virtual operating system environment a program which operates on data that is not very local in nature will have delays caused by page faulting. The conversion part of the program demonstrates these delays because of the large buffer and inherent nonlocal nature of accessing it. Another reason the program is slow is that the compression part of the algorithm is very CPU intensive since it is required to do byte level comparisons of the data with the previous raster line as well as doing checks along a raster line for repeating bytes.

The investigations to be discussed in this paper cover only the conversion portion of the algorithm, that of converting vector data to raster data. The compression portion will be presented at a future time.

VECTOR TO RASTER CONVERSION

The process of converting vector data to raster data "merely" requires determining which bits of a bit buffer (raster buffer) to turn on. There are several algorithms available for deciding which bits to turn on given the (X,Y) endpoints of a vector. The bit buffer in the AP uses 32,768 MD data words. It is configured as an array 32 by 1024 words. Each word uses only 32 of the 38 possible bits. Ideally the right-most 32 bits would be used. However, as will be discussed, this is not possible. One of the constraints in turning on a bit in the buffer is that no other bits be affected, i.e. one, and only one, bit location should be turned on per access. This is accomplished in a more conventional memory by inclusive ORing a 1 into the memory buffer word at the appropriate bit location. Unfortunately the AP does not allow for MD size (38 bits) word logical operations (OR, XOR, AND) or shifts or rotates. These functions can be approximated by using an existing set of operations in a nonconventional manner. This requires combinations of SPAD operations, OR and MOVR (rotate), floating point operations, FSCLT (floating scale truncate which approximates a rotate) and FOR (floating OR). The AP hardware has been designed so that normalization cannot be turned off. The implication is that if the result of the FOR instruction is not normalized it will be forced to be normalized, which causes the bits to be rotated in the mantissa, very undesirable when the specific bits to be on or off are important and the floating point value is not important. Another feature of the hardware is that the programmer has only limited control of convergent rounding. If the residue bits are set for some reason, the result will, in general, be convergently rounded. The FSCLT (floating scale truncate) instruction allows trucated rounding (i.e. no rounding) to occur but the FOR instruction will always round. The result of this discussion is that in order to avoid the problems with rounding and normalization, always set what we will call the "normalize bit". This is the least significant mantissa bit exclusive of the sign bit.

Another potential problem arises if it is desired to treat the sign bit as an useful data bit. To avoid any problem it was decided to always make the floating point number look positive, i.e. the sign bit always equals zero. This leads one to use only the right most 26 bits of the mantissa. Since each word must contain 32 bits, we are required to use 6 bits of the exponent field for the remaining bits. Since the exponent is 10 bits wide, using 6 of them does not pose any particular problem. The most convenient way to make sure the sign bit is always off and the "normalize bit" is always on is to preload the array with a floating point Ø.5 value. So when it is time to initialize the array (i.e. clear or erase it) for a new frame, we load the array with Ø.5. More particulars on the implementation will be deferred until later. What is established now is the fact that we can have 32 bits per words which can be treated as individual bit addresses.

TURNING ON THE BITS

We now develop an AP routine which we will call FILLAP. It is called with two parameters, the X and Y bit address, each of which runs from Ø to 1Ø23. The Y address can be used directly to generate a row address in the two-dimensional array buffer /FIELD(32.,1Ø24.)/. The X value is divided into two parts. The high 5 bits generate a column address into the array and the low 5 bits specify which one of the 32 bits of the addressed word to turn on. Having located the correct word in MD, and having generated the constraints on which bits in the word it is most convenient to use, we will get down to the business of actually getting the bit turned on.

From the previous discussion, we noted that there are two cases to consider. The first case is when the desired bit is one of the twenty six bits in the mantissa. The second case is when the desired bit is one of the six bits in the exponent. The first case is handled by scaling a 1.Ø to position the "on" bit to the appropriate location and then associating with the resulting mantissa the exponent from the buffer memory word. The exponent of the buffer memory word must be used to prevent the floating adder unit from repositioning the bit to be ORed when aligning the exponent fields. The buffer memory word and scaled 1.Ø (with its appropriate exponent) are then FORed together and the result is stored back into memory.

The second case is handled by treating the exponent as a separate integer field that can be accessed in the MD. A note of caution here: when transferring between MD and the SPADs to or from the exponent field, an exponent bias is added or removed. Since the exponent field is ten bits wide, the bias, for our purposes, can be ignored for the most significant six bits. Realizing this, we just read the exponent portion of the memory word into an SPAD. To turn on the appropriate bit a loop is set up to rotate a 1 into the correct position, then the two SPADs are ORed together. The result is stored into the exponent field in a data pad register and the entire new word is stored back into the buffer memory.

TIMING TEST AND RESULTS

The subroutine which contains the algorithm for determining where to place the dots when converting from vectors to rasters is called DRAW. It is written in FORTRAN and was compiled with APFTN (the FPS FORTRAN cross-compiler). DRAW is a completely integer routine and uses very simple standard FORTRAN. The timing of DRAW, exclusive of calls to FILLAP, are given for slow memory in AP clock cycles. This equation is only an average since there are several branches in DRAW. "N" is the length of the resulting vector in bits. Time=148+59N cycles per vector. Each on-bit generates one call to FILLAP which (on average) requires 36 cycles. The aggregate result is 148+95N cycles per vector. A sample test plot was generated which has 6 1024-bit long vectors and one each of 2, 4, 8, 16, 32, 64, 128, and 256 bit long vectors. This gives an average of 634202 cycles or 106 milliseconds (ms). The PDP-10 required 458ms to accomplish the same task yielding a performance ratio of 4.3(AP) to 1 (PDP10). This is not the factor of six improvement that we might have liked to see. However, this speed enhancement is sufficient to deem the endeavor a success. The portion of the conversion yet to be done is the compression. This is expected to gain at least the same factor of four because each row can be broken down into its bytes and then comparison can go on in the SPADs.

HASI MODIFICATIONS

The modifications to the Host AP Software Interface (HASI) were limited to the routine REBACK (see appendix). For test purposes it was necessary to read the bit buffer back to the PDP-10. Two simple changes had to be made to the HASI to accommodate this. The PDP-10 common block was called BUFXX and actually contains more variables than just FIELD. By simply replacing the common block generated in the HASI with the appropriate name and variables the correct linkage was obtained. The second change was required to allow long integers to be transfered from the AP to the PDP-10. The only format recognized by APAL for integers is 16 bit integers. APEX and the interface will actually allow long (full PDP word) integers. By changing the format type generated in the APGET call in the HASI we were able to transmit back the 36 bit integers.

CONCLUSIONS

Whereas doing long (32bit) integer arithmetic is very cumbersome on the AP, the manifest speed difference between the AP cycle time and the PDP-10 average cycle time is sufficient to allow consideration of the AP for non-floating point tasks.

REFERENCES

1. B. J. Heath, "Electrostatic Plotter Raster Construction Using the AP-120B", Proceeding of the FPS Users' Group 1979 pp116-132.

APPENDIX

The following appendix contains a listing of the APAL code used to link to the AP FORTRAN, clear the buffer (ERASE), read the finished buffer back to the host (REBACK), and actually turn the bits on (FILLAP). REBACK actually does nothing useful for the AP data but instead provides a way for the HASI to get access to the AP data buffer.

```
        $TITLE REBACK
"FORCE A READ BACK OF THE BUFFER TO THE PDP-10
"CODED BY NEIL MARON  19FEB80 LLL
        $SUBR REBACK,0
        $COMIO BUFF 1   "AP TO HOST ONLY
        $COMMON /BUFF/ FIELD(32.,1024.)/R
REBACK: RETURN
        $END
        $TITLE FILLAP   "(IX,IY)
"FILL AP MEMORY WITH A 1 BIT AT (IX=0-1023)(IY=0-1023)
"CODED BY NEIL MARON  24OCT79 LLL
        $SUBR FILLAP,2
        $COMIO BUFF 0   "NO TRANSFER
        $PARAM 2, AIX, AIY
        $COMMON /.LOCAL/ AIX,AIY
        $COMMON /BUFF/ FIELD(32.,1024.)/R
"SPAD
        T0      $EQU    0
        T1      $EQU    1
        IX      $EQU    2
        IY      $EQU    3
        FXY     $EQU    4
        IXW     $EQU    5
        IBIT    $EQU    6
FILLAP: LDMA; DB=AIX            "GET IX
        LDMA; DB=AIY            "GET IY
        NOP
        LDMA; DB=MD
        LDMA; DB=MD
        NOP
        LDSPI IX; DB=MD
        LDSPI IY; DB=MD
"IXW=IX/32=IX SHIFT RIGHT 5
        MOVR IX,IXW
        MOVRR IXW,IXW
        MOVRR IXW,IXW
"IBIT=IX .AND. 37(8) =>PICKING LOW 5 BITS
        LDSPI IBIT; DB=37         "GET THE 5BIT MASK
        AND IX,IBIT               "DO THE AND
"GET FXY=<FIELD(IXW,IY)>=FIELD+IY*32+IXW=(IY SHIFT LEFT 5) + IXW + FIELD
        MOVL IY,FXY      "MOVE TO DESTINATION AND DO FIRST SHIFT
        MOVL FXY,FXY
        MOVL FXY,FXY
        MOVL FXY,FXY
        MOVL FXY,FXY
        ADD IXW,FXY               "NOW ADD IXW
        LDSPI 1;DB=FIELD          "GET VALUE OF FIELD
        ADD 1,FXY                 "NOW ADD FIELD AND WE ARE DONE
"NOW BRANCH TO THE APPROPRIATE SECTION OF CODE
"FOR IX=Ø TO 31 AND APWORD= (1ØEXP)(1SIGN)(27MANT) WE MAP
"AS FOLLOWS: (THINGS IN [] ARE ACTUAL VALUES, UN[]-ED NUMBERS ARE BIT POS.
" [X][X][X][X]Ø12345[Ø][1]6..31     = FIELD WORD CONTENTS
"  Ø  1  2  3 456789                = EXPONENT
"                    Ø  1 2..27     = MANTISSA FIELD
"NOTES:  SIGN IS ALWAYS OFF=POSITIVE AND NORMALIZE BIT IS ALWAYS ON(=1)
" CASE 1 FOR IBIT=6 TO 31
" CASE 2 FOR IBIT=Ø TO 5
        LDSPI TØ; DB=5  "CHECK FOR Ø-5
        SUB# IBIT,TØ
        BGE CASE2; LDSPI T1; DB=3  "JUST IN CASE ITS CASE 1 WE NEED A 3
CASE1:  SUB T1,IBIT;              "SET FOR FSCLT INSTR
         LDTMA; DB=!ONE           "GET A 1.0 TO SCALE
        MOV# FXY,FXY; SETMA       "GET WORD
        NOP
        DPX<TM
        DPY<MD; MOV# IBIT,IBIT; FSCLT DPX
        FADD; LDSPE TØ; DB=DPY  "GET EXP FROM MEM WORD
        DPX<FA          "STORE NORMALIZED BIT INTO DPX
        FOR DPY,MDPX; MOV# TØ,TØ          "OR TOGETHER USING MEM EXPON
        FADD                              "PUSH
        MOV FXY,FXY; SETMA; MI<FA         "DONE, PUT RESULT INTO MEM
        RETURN
CASE2:  MOV FXY,FXY; SETMA        "GET MEM WORD
        LDSPI T1; DB=1ØØ          "BIT TO TURN ON
        MOV IBIT,IBIT
C2LOOP: BEQ C2OKAY; MOVR T1,T1   "SPARE ROT IS OKAY-DONT GET MD YET
        DEC IBIT; BR C2LOOP
C2OKAY: LDSPE TØ; DB=MD; DPX<MD "GET EXPON AND SAVE MEM IN DPX
        OR T1,TØ                  "OR BITS
        DPX<SPFN; WRTEXP          "WRITE NEW EXPONENT ONLY
        MOV FXY,FXY; SETMA; MI<DPX        "STORE NEW WORD
        RETURN
        $END
"
"ENTRY TO ERASE THE FIELD
"
        $TITLE ERASE
        $SUBR ERASE,Ø
        $COMIO BUFF Ø   "NO DATA TRANSFER
        $COMMON /BUFF/ FIELD(32.,1Ø24.)/R
```

```
ERASE:  LDSPI 0; DB=FIELD-1       "INIT START ADDR
        LDTMA; DB=!HALF           "GET A .5 TO STORE
        LDSPI 1; DB=32768.        "INIT LENGTH
ERASEL: DEC 1
        INC 0; MI<TM; SETMA; BGT ERASEL
        RETURN
        $END
```